

FIG. 1

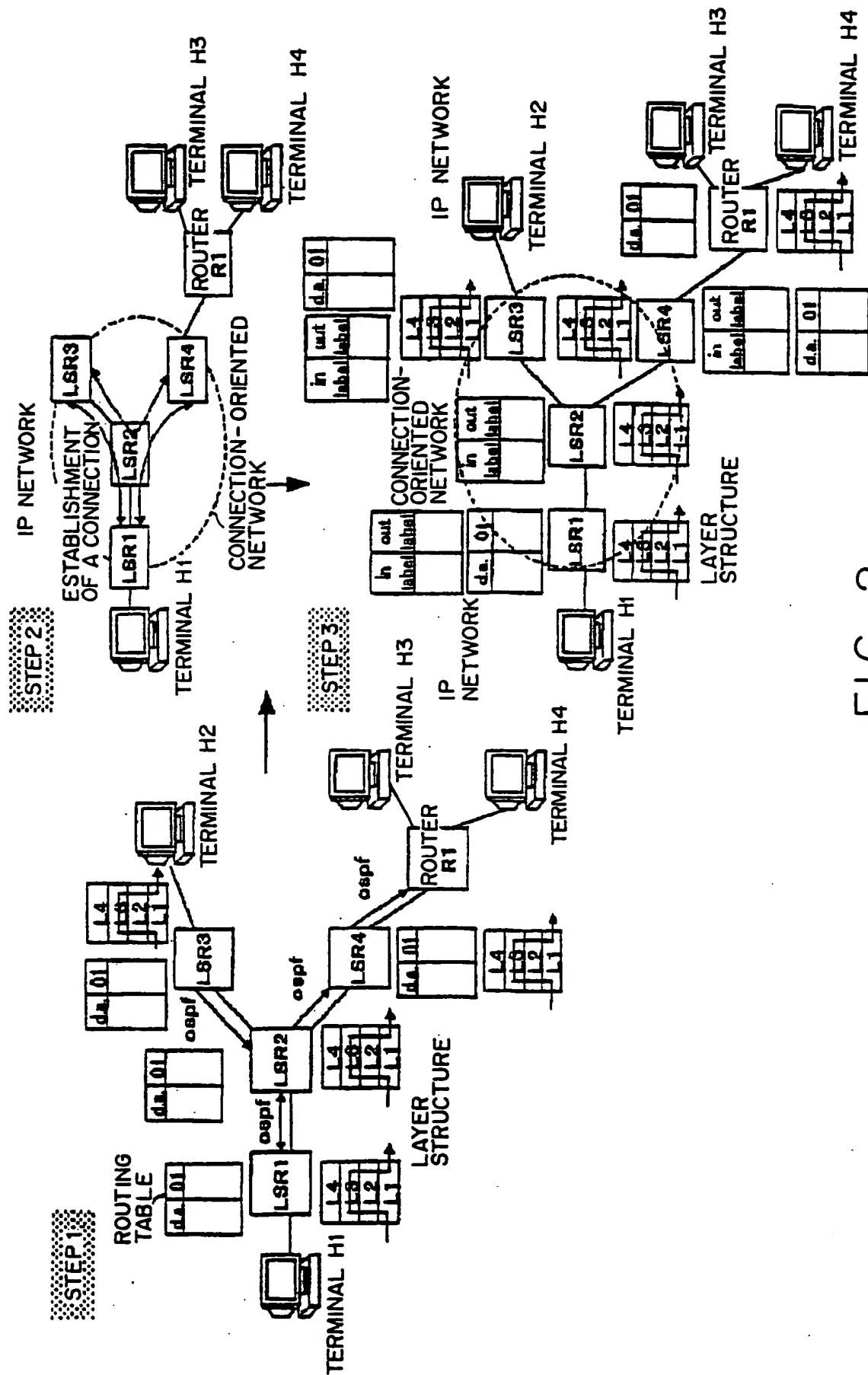


FIG. 2

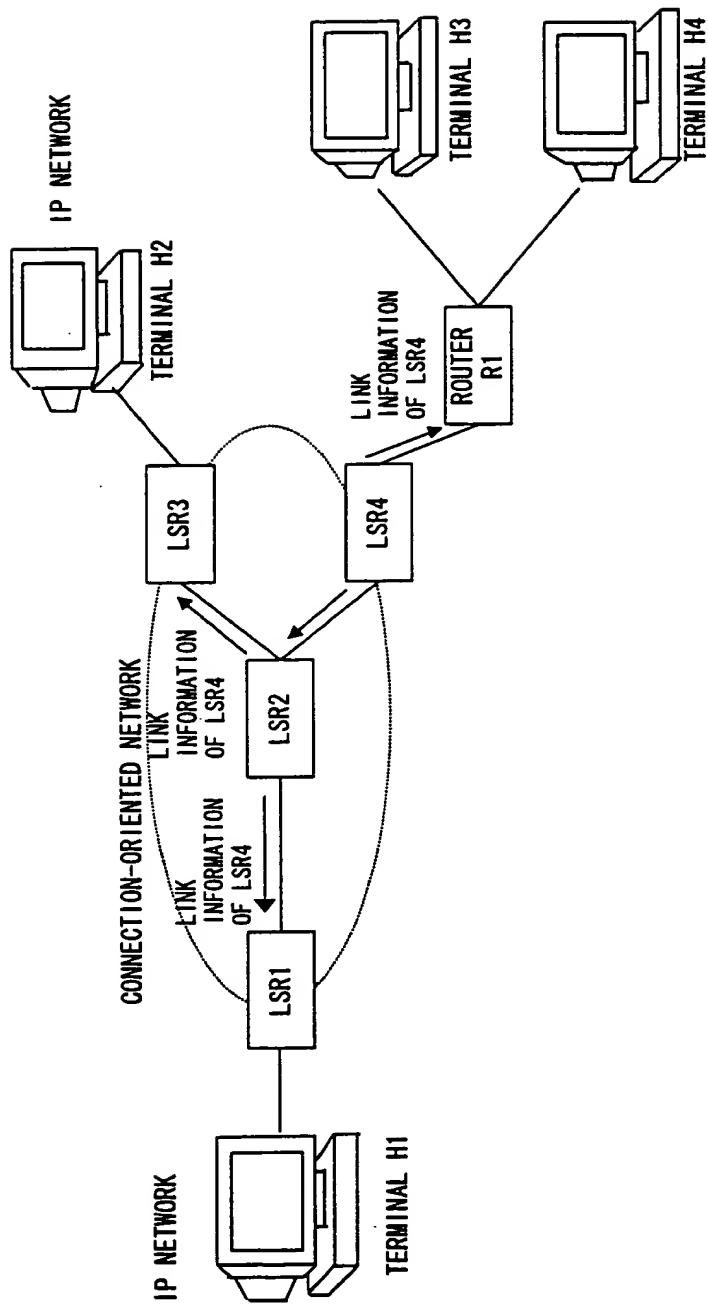


FIG. 4

| | | | | | | | |
|---|---|----|----|-----|----|---|---|
| * | O | DC | EA | N/P | MC | E | * |
|---|---|----|----|-----|----|---|---|

THE OPTIONS FIELD

F I G. 5 A

| | | | | | | | |
|---|---|----|----|-----|----|---|---|
| L | O | DC | EA | N/P | MC | E | * |
|---|---|----|----|-----|----|---|---|

THE OPTIONS FILED

F I G. 5 B

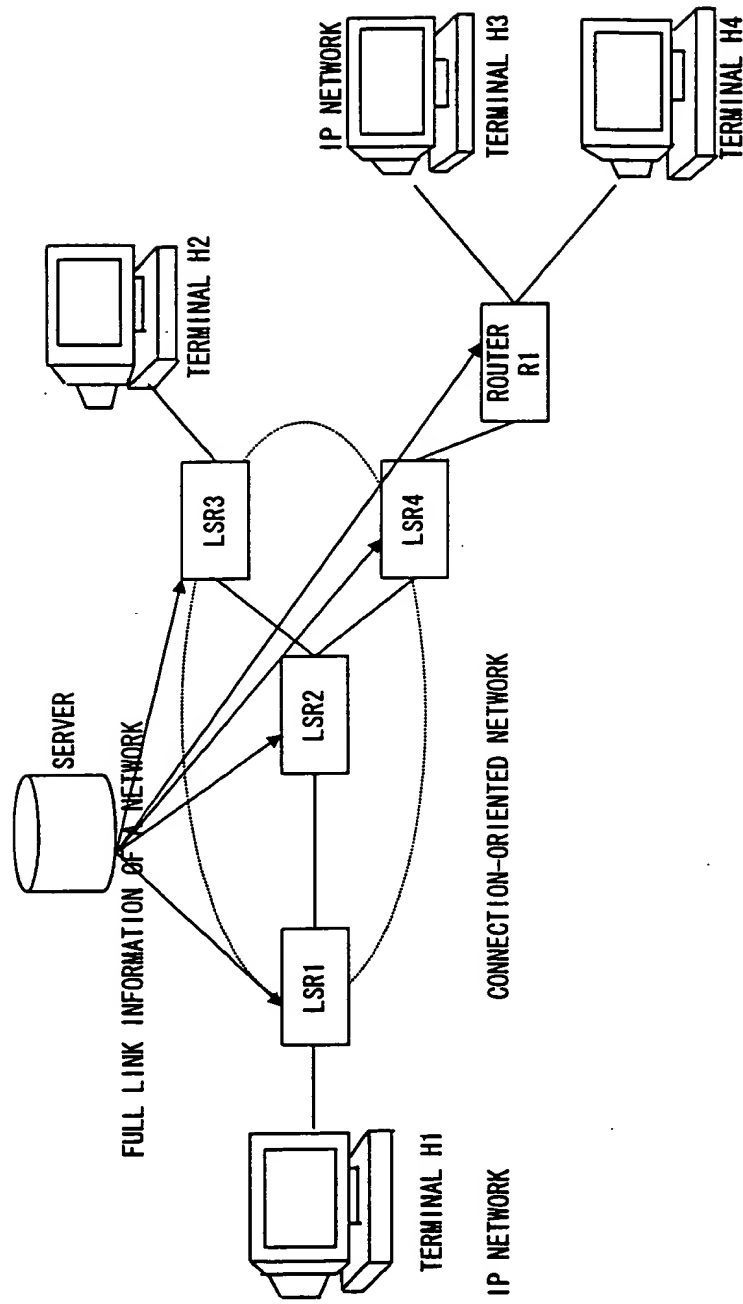
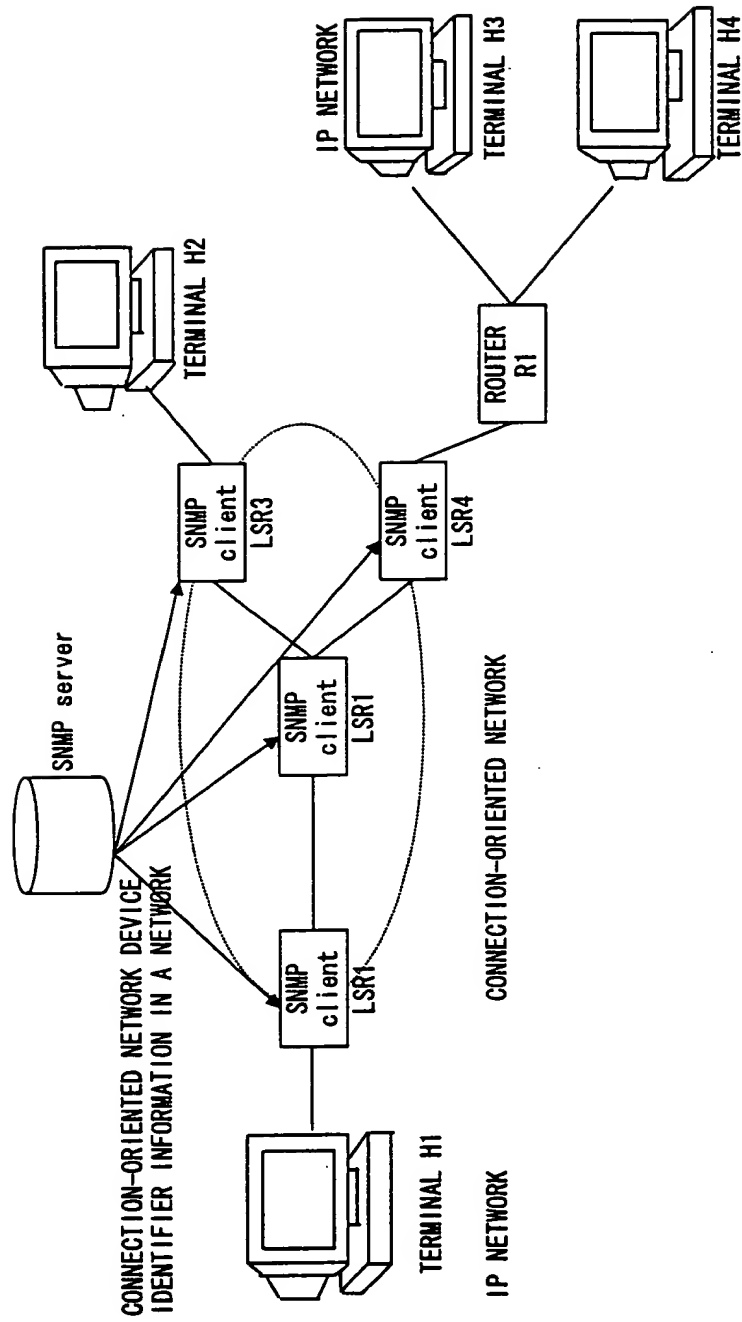


FIG. 6



009227 62464260

| | | | | | | | |
|---|---|----|----|-----|----|---|---|
| * | 0 | DC | EA | N/P | MC | E | R |
|---|---|----|----|-----|----|---|---|

THE OPTIONS FIELD

F I G. 8

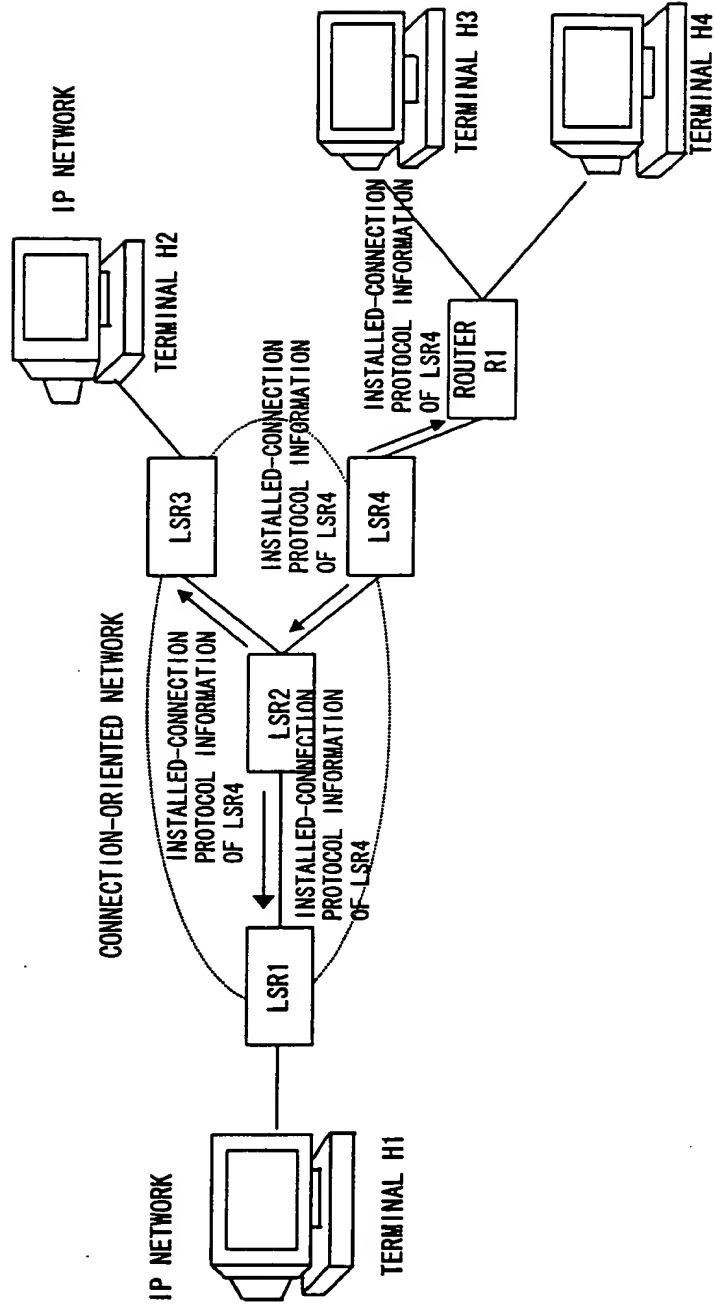


FIG. 9

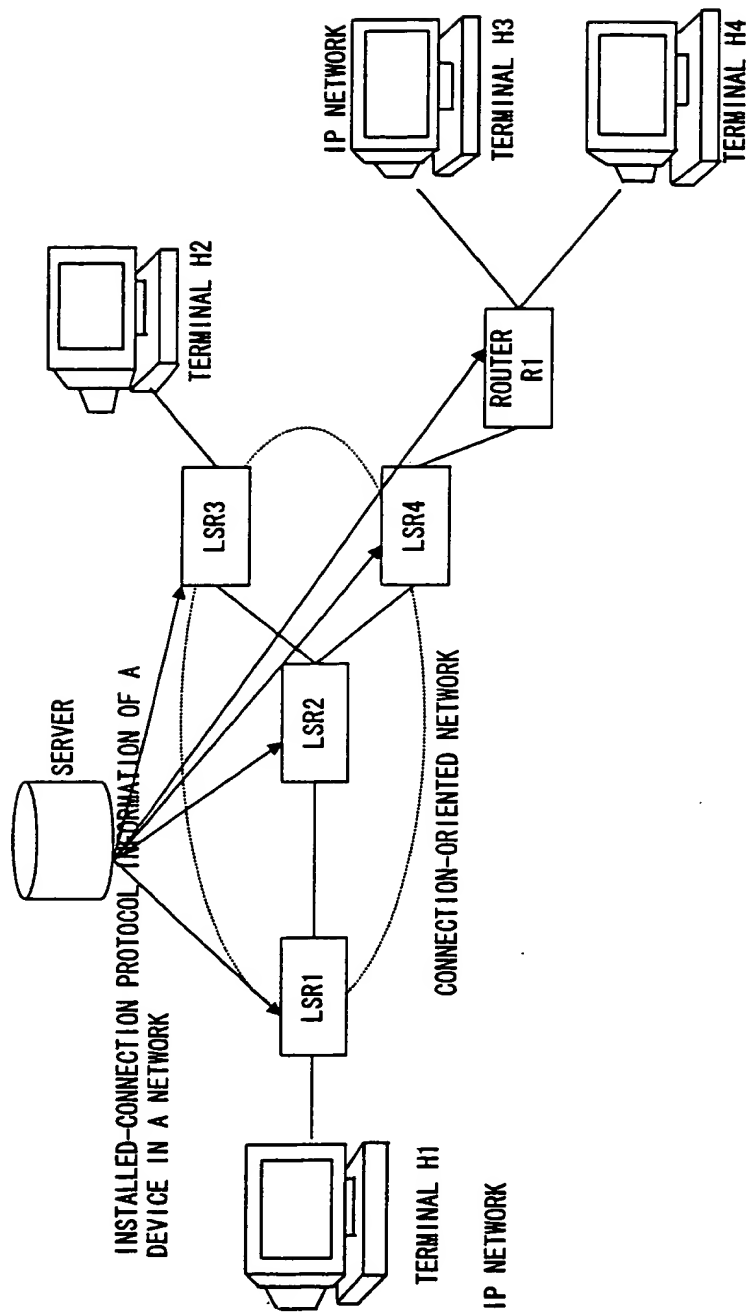


FIG. 10

009227" 64767250

| DEVICE (INTERFACE) ADDRESS | CONNECTION-ORIENTED NETWORK DEVICE IDENTIFIER | CONNECTION PROTOCOL IDENTIFIER |
|----------------------------|--|--------------------------------|
| 10.0.0.1 | O | O |
| 10.25.1.1 | O | x |
| ... | ... | ... |

FIG. 11

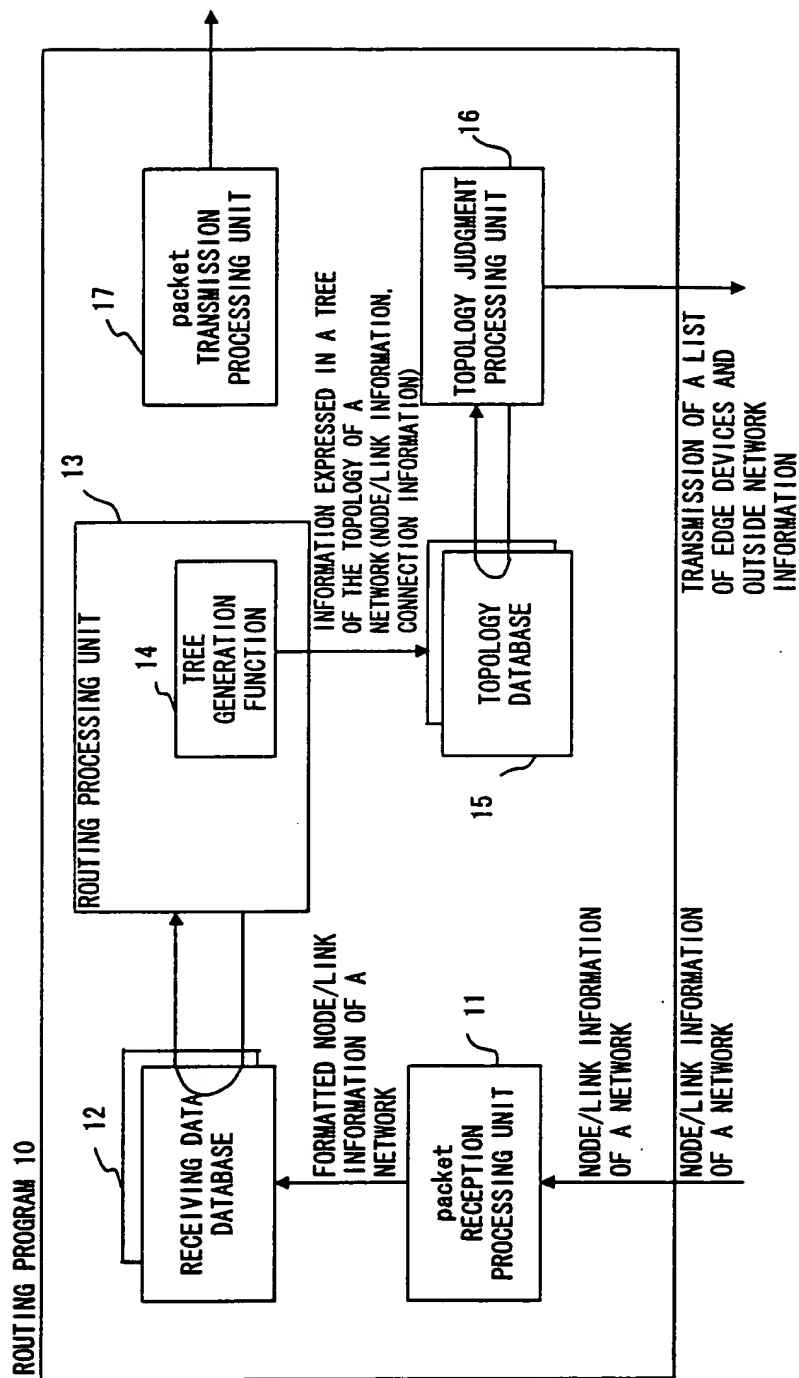


FIG. 12

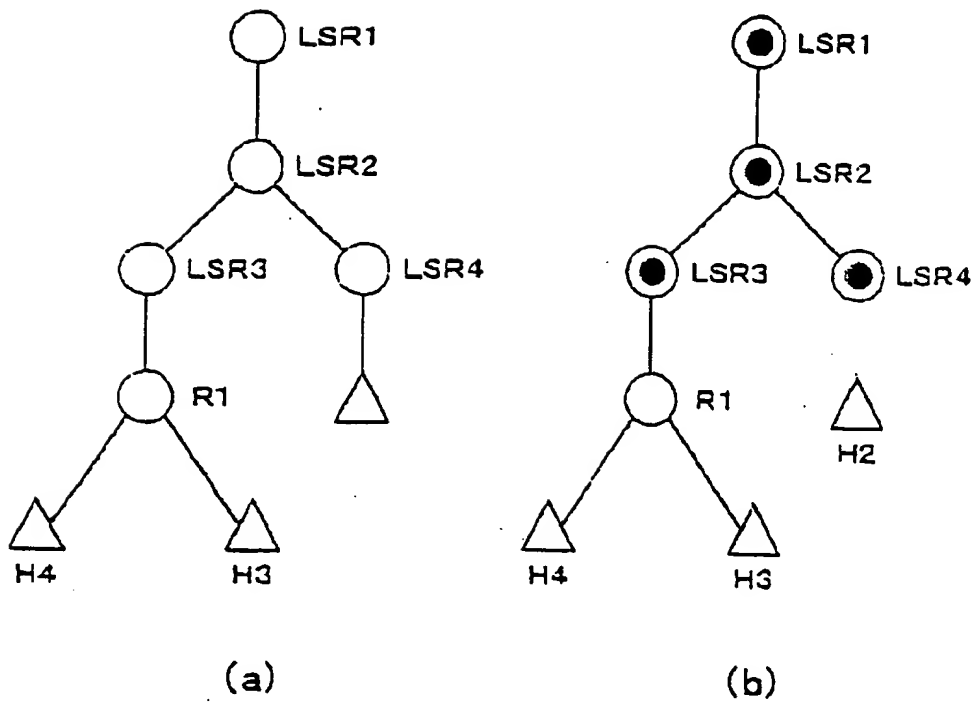
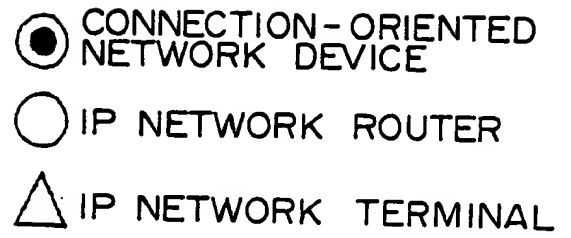


FIG. 13

```

/*Initialization*/
DESIGNATES A SELF-NODE AS THE ROOT OF A TREE.
current_pointer=SELF NODE
/*spf routine from here*/
spf_routine()
[
    while(1)[
        SEARCHES FOR LINK INFORMATION ADJACENT (RELATED) TO current_pointer
        IF (A NEW ENTRY IS DETECTED.)[
            ADDS THE NEW ENTRY TO THE TREE.
            IF (L/R BITS OF THE LINK INFORMATION OF AN ENTRY "option header" BOTH ARE ON) [ ..... (1)
                THE DEVICE IS A VALID CONNECTION-ORIENTED NETWORK DEVICE
                (THE INFORMATION IS INTERNALLY STORED).
            ]
            current_pointer=NODE OF A NEW ENTRY
        ]
        IF (THERE IS NO ENTRY.)[
            current_pointer=ONE-RANK HIGHER NODE
            IF (current_pointer=null)[
                return(0)+ /*THE SEARCH IS COMPLETED. EXITS THE ALGORITHM. */
            ]
        ]
    ]
]

```

FIG. 14

```

/*Initialization*/
int traced []
/*CHECKS WHETHER ALL POSITIONS ARE CHECKED (CHECKED=1,
UNCHECKED=0)*/
int *current_pointer=LSR1 /*INITIALIZES THE CURRENT POSITION IN THE TREE.*/
char edge_entry [] /*EDGE DEVICE ENTRY*/
int edge_entry_number /*TOTAL NUMBER OF EDGE DEVICE ENTRIES*/

/*SEARCH ROUTINE*/
search() {
    while(1) {
        /*VIEWS THE CHILD OF A DEVICE POINTED TO BY current_pointer.
        IF (traced[child]=0) {
            current_pointer=child; /*MOVES TO AN UNCHECKED CHILD*/
            traced[child]=1;
        }
    }

    ELSE IF (ALL CHILDREN ARE CHECKED.) {
        current_pointer=parent /*MOVES TO THE PARENT*/
        IF (parent=null) {
            break; /*THE SEARCH IS COMPLETED.*/
        }
        continue;
    }

    /*JUDGES WHETHER IT IS AN EDGE DEVICE.*/
    IF (L BIT POINTED TO BY current_pointer is zero.) {
        edge_entry[edge_entry_number]=parent DEVICE;
        ++edge_entry_number;
        current_pointer=parent;
    }
}

return(0);

```

FIG. 15

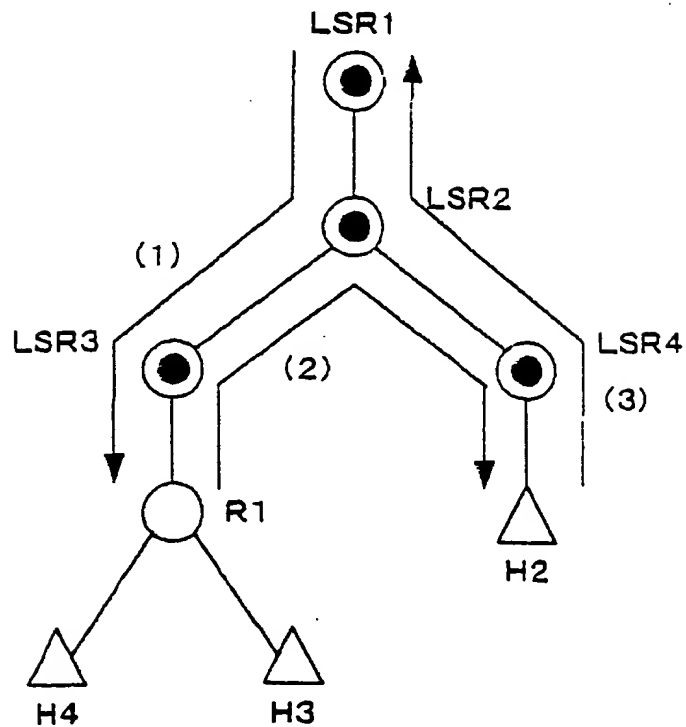
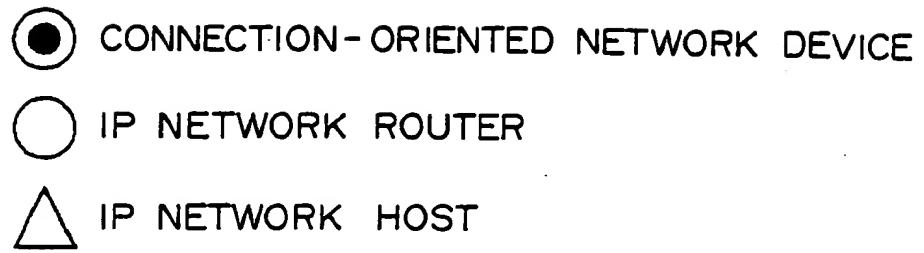


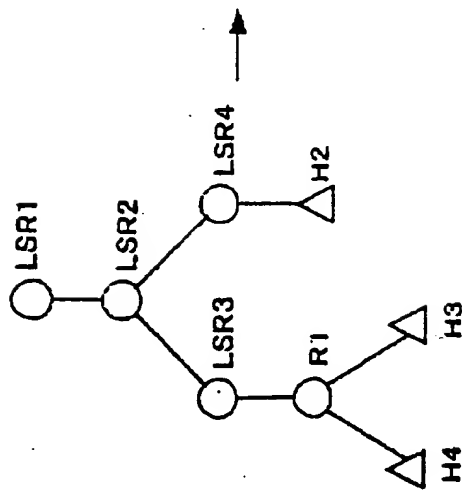
FIG. 16

009227" 62464260

EDGE DEVICE ENTRIES STORED IN LSR1

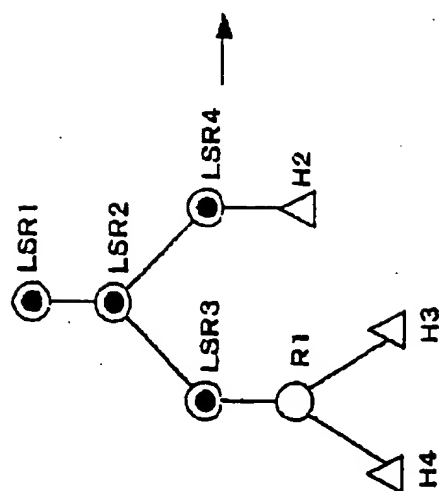
| EDGE DEVICE |
|-------------|
| LSR3 |
| LSR4 |

F I G. 1 7



| destination address(d.a.) | next hop outgoing Interface(OI) |
|------------------------------|------------------------------------|
| | |

FIG. 18A



| destination address(d.a.) | egress LSR IP address | next hop (outgoing Interface(OI)) |
|------------------------------|--------------------------|--------------------------------------|
| | | |

FIG. 18B

```

/*initialization*/
int traced [] /*CHECKS WHETHER ALL POSITIONS ARE CHECKED (CHECKED=1,UNCHECKED=0)*/
int *current_pointer=LSR1 /*INITIALIZES THE CURRENT POSITION IN THE TREE.*/
char edge_entry [] /*EDGE DEVICE ENTRY*/
int edge_entry_number /*TOTAL NUMBER OF EDGE DEVIE ENTRIES*/
/*SERCH ROUTINE*/
search() {
    while(1){
        VIEWS THE child OF A DEVICE POINTED TO BY current_pointer.
        IF(traced[child]=0) {
            current_pointer=child: /*MOVES TO AN UNCHECKED CHILD.*/
            traced[child]=1:
        }
        ELSE IF(ALL CHILDREN ARE CHECKED.) {
            current_pointer=parent /*MOVES TO THE parent/*
            IF(parent=null) {
                break: /*THE SEARCH IS COMPLETED.*/
            }
            continue:
        }

        /*JUDGES WHETHER IT IS AN EDGE DEVICE.*/
        IF(L BIT POINTED TO BY current_pointer IS ZERO.) {
            edge_entry[edge_entry_number]=parent DEVICE:
            ++edge_entry number:
        }

        /*ADDS AN ENTRY OF A NETWORK CONNECTED TO AN EDGE DEVICE.*/
        IF(L BIT POINTED TO BY current_pointer IS ZERO.) {
            RELATES THE ip address POINTED TO BY curren_pointer TO
            edge_entry[edge_entry_number](ENTRY ADDITION);
        }
    }
    return(0):
}

```

F I G. 1 9

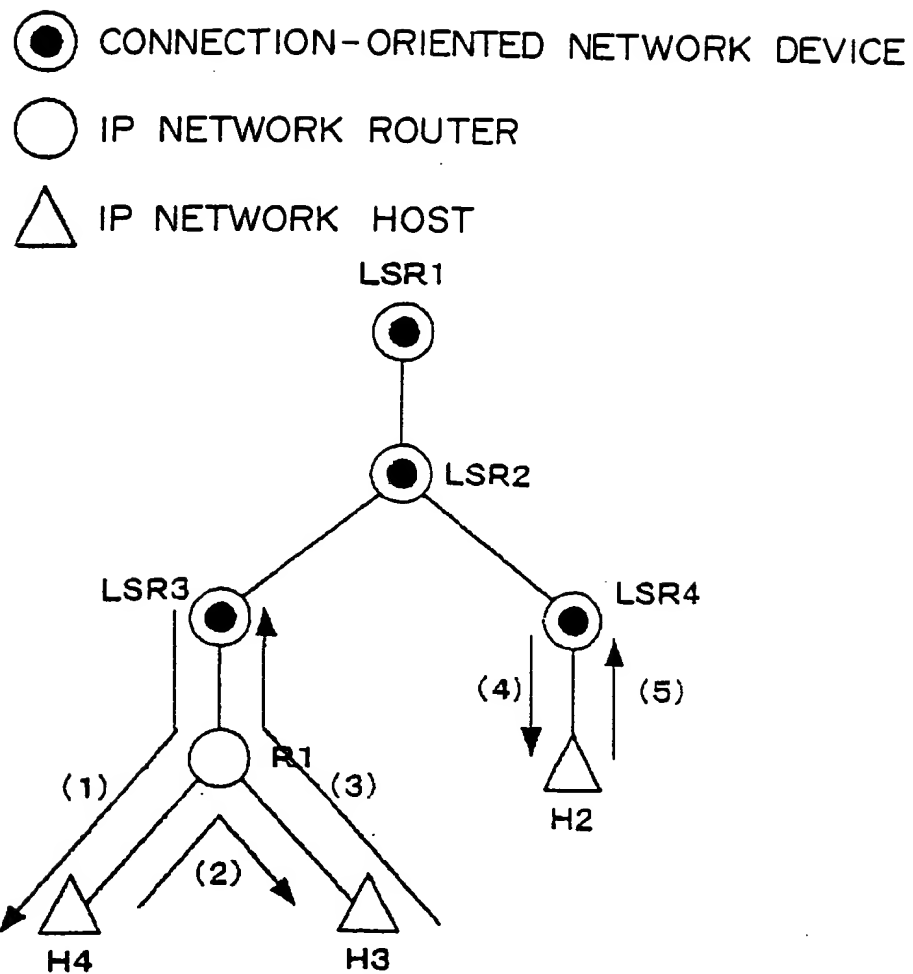


FIG. 20

ENTRY OF EDGE DEVICE/OUTSIDE NETWORK INFORMATION STORED
IN LSR1

| EDGE DEVICE | OUTSIDE NETWORK |
|-------------|-----------------|
| LSR3 | R1 |
| LSR3 | H4 |
| LSR3 | H3 |
| LSR4 | H2 |

F I G. 2 1

00522T" 02464460

routing_table_request object

OBJECT INSERTED IN A PATH MESSAGE. IF SENDER(ENTRANCE EDGE DEVICE) WANTS TO OBTAIN THE ROUTING
TABLE OF AN EXIST EDGE DEVICE, routing_table_request object IS INCLUDED IN THE PATH MESSAGE.

routing_table object

ON RECEIPT OF THE PATH MESSAGE, INCLUDING THE routing_table_request object, AN EXIT EDGE DEVICE RETURNS
AN RESV MESSAGE, INCLUDING THE routing_table object, TO THE SENDER. THE file OF THE routing table IS COPIED
INTO THE routing_table object AND IS TRANSMITTED.

F I G. 2 2

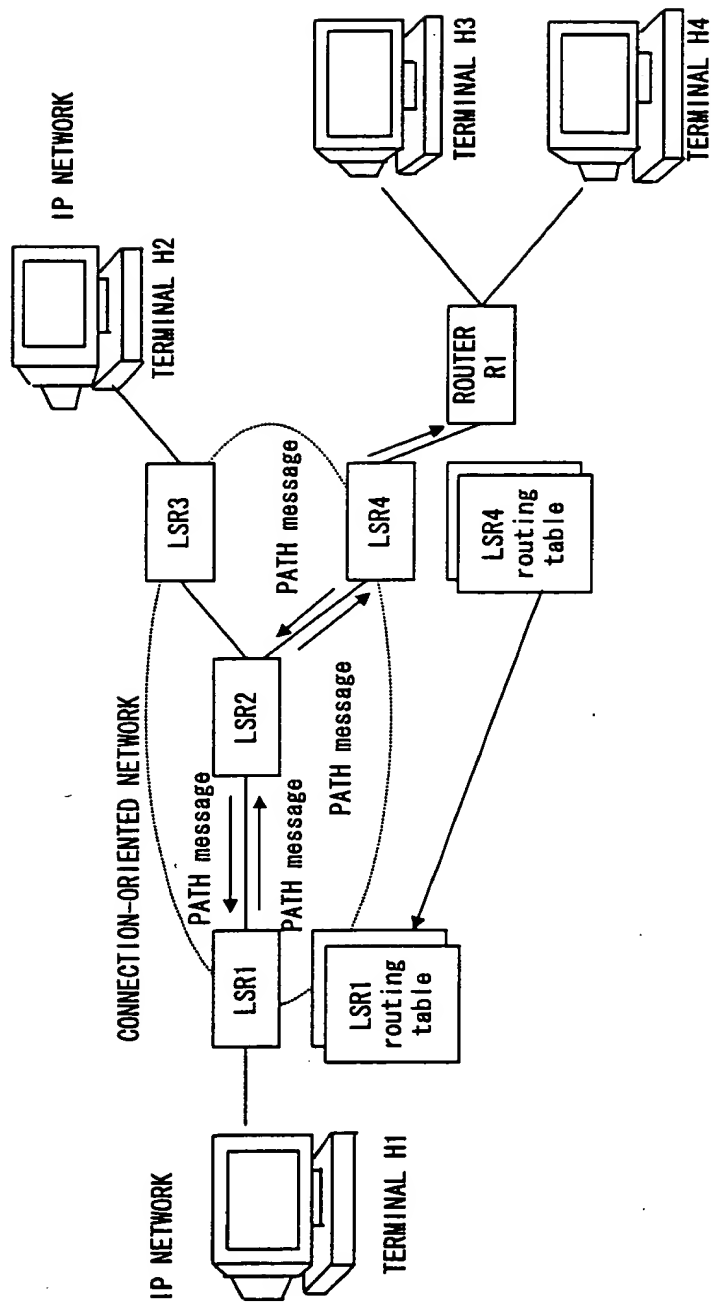


FIG. 23

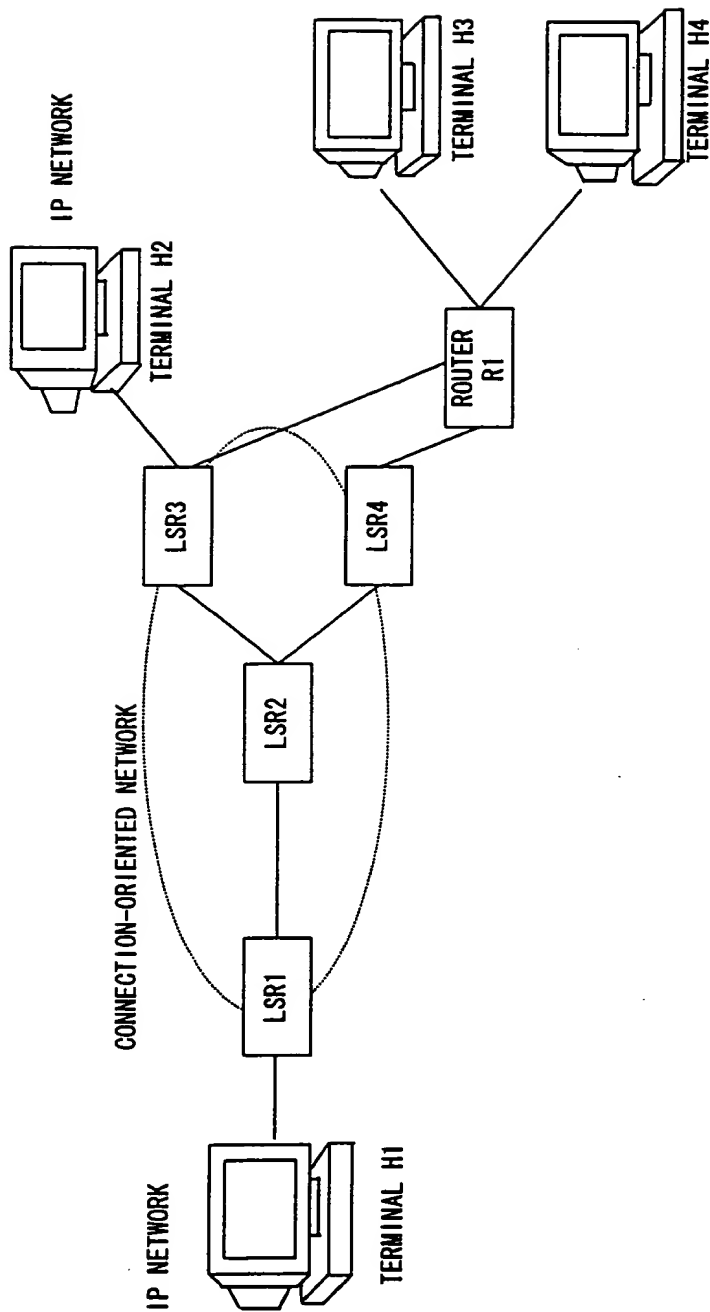


FIG. 24

00922F" 62464260

| FEC | | | | | label |
|----------|---------|---------|---------|---------|-------|
| d. a. | s. a. | d. p. | s. p. | proto | |
| 10.0.0.1 | | 1000 | | | 50 |
| 10.0.0.1 | | 1050 | | | 60 |
| 20.0.0.0 | | | | | 100 |

d.a. =destination IP address, s.a.=source IP address
d.p. =destination port, s.p.=source port
proto=protocol ID, ...=no-designation

F I G. 2 5

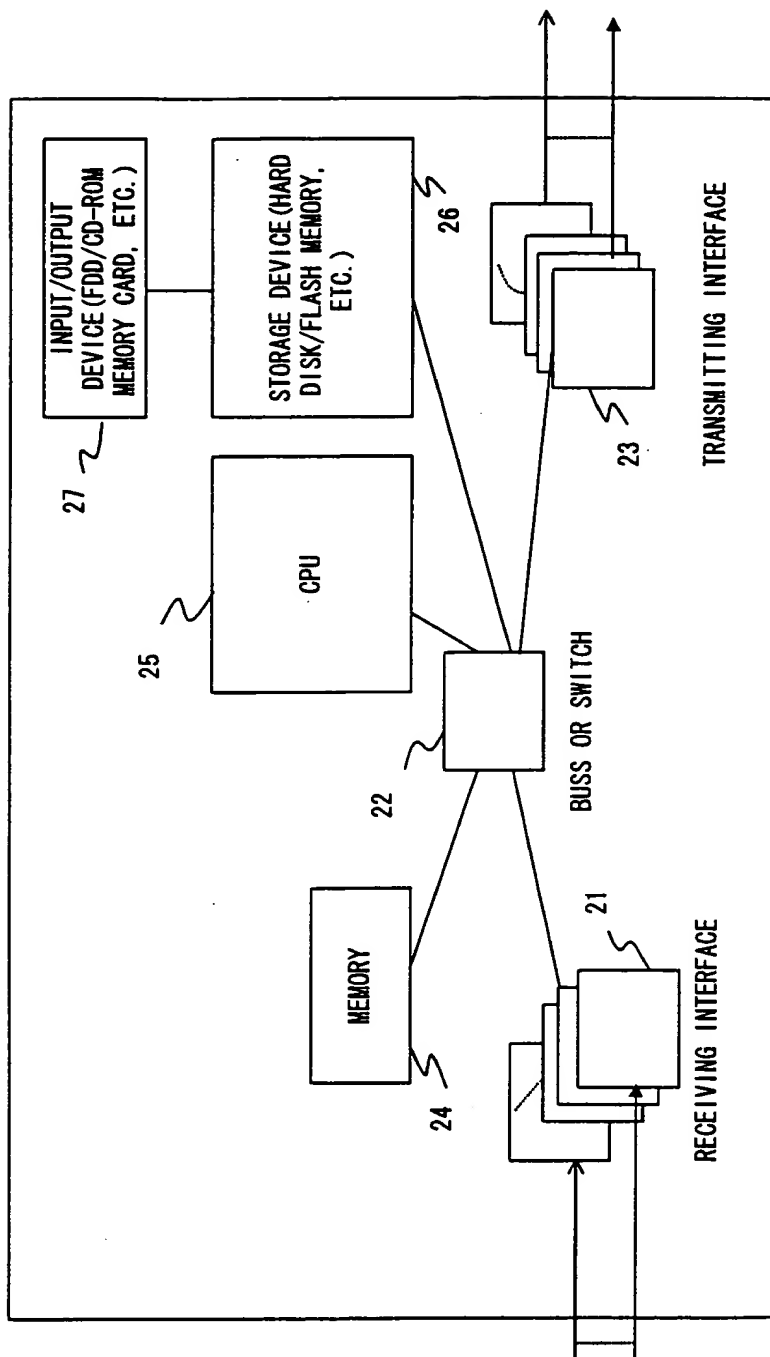


FIG. 26